

Basic EPICS Device Support

COLLABORATORS

	<i>TITLE :</i> Basic EPICS Device Support		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Michael Davidsaver	April 2009	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME
1	April 2009	Initial revision	MAD

Contents

1	Introduction	1
2	Prepare IOC environment	1
3	PRNG Example	1
3.1	Device Definition	2
3.2	Writing Support	2
3.3	Building	3
3.4	Database Configuration	4
3.5	Running	4
4	Asynchronous Example	4
4.1	Running	7
5	Now What	7
6	References	7

1 Introduction

Device support is the means of providing functionality specific to access hardware. This is done by providing several functions which the record support layer will call when appropriate. The functions which must be provided depend on the record type being supported. The *Record Reference Manual* [RecRef] provides a list of record types with descriptions and lists of device support functions. To determine the exact form of a record's device support functions the source found in `$EPICS_BASE/src/rec` is invaluable.

2 Prepare IOC environment

It is assumed that EPICS Base is already built, that `EPICS_BASE` is set, and that the EPICS executables are in the system search PATH.

All paths given in this example are assumed to be relative to the *devsumexample* directory.

```
$ mkdir devsupexample $ cd devsupexample
```

3 PRNG Example

Let us begin with an example. The Analog input record is intended to represent a value read from hardware and interpreted as a floating point number. This does not imply that the underlying hardware representation is a floating point number. The AI record support provides a facility for conversion between a raw integer value and a floating point number.

In this example the *hardware* device to be read is the system pseudo random number generator. Whenever the record is processed a new number is read into the process variable (PV) database.

```
makeBaseApp.pl -t ioc prng
```

This creates the makefiles needed to compile the code. The files we are about to create will be placed in *prngApp/src*. Later when *.db* files are created we will place them in *prngApp/Db*.

Take a moment to examine the files in *prngApp/src*. The file *prngMain.cpp* will be the point of entry for our IOC when run on a non-embedded platform. It is not very interesting through as it serves only to invoke the IOC shell.

The *prngApp/src/Makefile* does contain several interesting entries. Removing comments and blank lines leaves the following.

```
TOP=../..
include $(TOP)/configure/CONFIG
PROD_IOC = prng
DBD += prng.dbd
prng_DBD += base.dbd
prng_SRCS += prng_registerRecordDeviceDriver.cpp
prng_SRCS_DEFAULT += prngMain.cpp
prng_SRCS_vxWorks += -nil-
prng_LIBS += $(EPICS_BASE_IOC_LIBS)
include $(TOP)/configure/RULES
```

It is important to note that the EPICS build system attaches special significance to file names, not just extensions.

This Makefile will build the *prng* IOC (executable) from the two given C++ files and the database definition. Of these three only *prngMain.cpp* exists currently. The file *prng_registerRecordDeviceDriver.cpp* is automatically generated from the database definition. The database definition file *prng.dbd* is also generated by concatenating *base.dbd* with other *.dbd* files which we will add later. At this point is effectively just a copy of *base.dbd*, which is part of the EPICS Base package and specifies, among other things, the basic record types (see *\$EPICS_BASE/dbd/base.dbd*).

At this point the *prng* IOC can now be compiled, and the resulting executable can be run. However, it will not be capable of doing anything more then the *softIoc* executable. In fact they are functionally identical.

3.1 Device Definition

Our first task is to make an addition to the IOC database for our prng device. Create the file *prngApp/src/prngdev.dbd*:

```
device(ai, CONSTANT, devAiPrng, "Random")
```

This defines the device *devAiPrng* as support for an AI record with a CONSTANT input link named "Random". The name *devAiPrng* must be unique in the IOC. The combination of record type and name string must also be unique. It is convention that device support names should take the form *devXxYyyy* where *Xx* is the record type and *Yyyy* identifies the hardware to be supported.

3.2 Writing Support

Now create the file *prngApp/src/devprng.c* with the following sections.

```
#include <stdlib.h>
#include <epicsExport.h>
#include <dbAccess.h>
#include <devSup.h>
#include <recGbl.h>

#include <aiRecord.h>

static long init_record(aiRecord *pao);
static long read_ai(aiRecord *pao);

struct prngState {
    unsigned int seed;
};
```

Our device support code will be contained in the *init_record* and *read_ai* functions. Custom state information will be held in an instance of the *prngState* structure.

```
struct {
    long num;
    DEVSUPFUN report;
    DEVSUPFUN init;
    DEVSUPFUN init_record;
    DEVSUPFUN get_ioint_info;
    DEVSUPFUN read_ai;
    DEVSUPFUN special_linconv;
} devAiPrng = {
    6, /* space for 6 functions */
    NULL,
    NULL,
    init_record,
    NULL,
    read_ai,
    NULL
};
epicsExportAddress(dset, devAiPrng);
```

Now associate the name *devAiPrng* with our device support functions. This mechanism is a way of providing a set of functions which the record support will use in to perform certain functions.

It should be noted that AI record support requires *read_ai* to be specified, but *init_record* is optional.

From the prospective of object oriented programming the record support can be regarded as a base class. This device support inherits from the AI record support and provides definitions for two virtual methods. The AI record requires that *read_ai* be provided while *init* is optional. In this way *read_ai* functions as a pure virtual method.

The number of functions record support will look for and the meaning of these functions is determined by record support. For information on a specific record types see the *Record Reference Manual* [RecRef] and the source (ie `$EPICS_BASE/src/rec/aiRecord.c`).

```
static long init_record(aiRecord *pao)
{
    struct prngState* priv;
    unsigned long start;

    priv=malloc(sizeof(struct prngState));
    if(!priv){
        recGblRecordError(S_db_noMemory, (void*)pao,
            "devAoTimebase failed to allocate private struct");
        return S_db_noMemory;
    }

    recGblInitConstantLink(&pao->inp,DBF_ULONG,&start);

    priv->seed=start;
    pao->dpvt=priv;

    return 0;
}
```

This *init_record* function is called once for each record in the IOC database which uses DTYP *Random* (ie *devAiPrng* device support). It allocates space for the structure used to keep internal state, then parses the Constant input link string to get the initial seed value.

The input link can only by a CONSTANT link so there is no need to verify this.

```
static long read_ai(aiRecord *pao)
{
    struct prngState* priv=pao->dpvt;

    pao->rval=rand_r(&priv->seed);

    return 0;
}
```

Whenever a record using *devAiPrng* is processed *read_ai* is invoked. It simply invokes the thread-safe version of the *rand* function to supply a raw (integer) value.

3.3 Building

Now modify *prngApp/src/Makefile* to include the prng database and support code. Then go to the *devsupexample* directory and run *make*

```
TOP=../..
include $(TOP)/configure/CONFIG
PROD_IOC = prng
DBD += prng.dbd
prng_DBD += base.dbd
prng_DBD += prngdev.dbd # <- added
prng_SRCS += prng_registerRecordDeviceDriver.cpp
prng_SRCS += devprng.c # <- added
prng_SRCS_DEFAULT += prngMain.cpp
prng_SRCS_vxWorks += -nil-
prng_LIBS += $(EPICS_BASE_IOC_LIBS)
include $(TOP)/configure/RULES
```

3.4 Database Configuration

The next task is to create a IOC database which uses the *Random* device support. Place the following in *prngApp/Db/prng.db* and add it to the makefile *prngApp/Db/Makefile*.

```
record(ai, "$(P) ") {
    field(DTYP, "$(D) ")
    field(DESC, "Random numbers")
    field(SCAN, "1 second")
    field(INP, "$(S) ")
    field(LINR, "LINEAR")
    field(ESLO, 1e-9)
    field(EOFF, -1)
}
```

This will allow us to create several PVs generating random numbers. The combination of record type *ai* and the *DTYP* field are used to identify the correct device support. When instantiated *\$(P)*, *\$(D)*, and *\$(S)* will be replaced with the PV name, device support type (*Random*), and initial seed value. These will be specified later.

The fields *ESLO* and *EOFF* serve to define a linear scale to use when converting (integer) raw values to (floating point) engineering units.

Note: when changing *prngApp/Db/prng.db* remember to run *make* to update *db/prng.db*.

3.5 Running

In *devsupexample* create the IOC boot infrastructure to run the first example (*prng1*).

```
makeBaseApp.pl -a linux-x86 -i -t ioc -p prng prng1
```

In *iocBoot/iocprng1/st.cmd*:

```
< envPaths
cd ${TOP}
dbLoadDatabase "dbd/prng.dbd"
prng_registerRecordDeviceDriver pdbbase
# V Add this line V
dbLoadRecords ("db/prng.db", "P=test:prng,D=Random,S=324235")
cd ${TOP}/iocBoot/${IOC}
iocInit
```

Now run the IOC.

```
make
cd iocBoost/iocprng1
../../bin/linux-x86/prng st.cmd
```

Then watch the value of the PV *test:prng*

```
$ camonitor test:prng
test:prng          2009-02-21 15:29:15.364549 0.155918
test:prng          2009-02-21 15:29:16.364611 -0.681225
...
```

4 Asynchronous Example

The preceding example assumes that calls to *read_ai* will return quickly. This is true of *rand_r* which does only a simple computation, but not true of many operations which access hardware. In these cases it is desirable to start an operation, spend time doing other things, and only update the database when the result becomes available.

Support for this mode of operation is provided via the *PACT* flag. The following example creates another device support which demonstrates asynchronous processing.

Add the following line to *prngApp/src/prngdev.dbd*

```
device(ai,CONSTANT,devAiPrngAsync,"Random Async")
```

Now create the file *prngApp/src/devprngasync.c* and add the following sections.

```
#include <stdlib.h>
#include <epicsExport.h>
#include <dbAccess.h>
#include <devSup.h>
#include <recSup.h>
#include <recGbl.h>
#include <callback.h>

#include <aiRecord.h>

static long init_record(aiRecord *pao);
static long read_ai(aiRecord *pao);

struct prngState {
    unsigned int seed;
    CALLBACK cb; /* New */
};

struct {
    long num;
    DEVSUPFUN report;
    DEVSUPFUN init;
    DEVSUPFUN init_record;
    DEVSUPFUN get_ioint_info;
    DEVSUPFUN read_ai;
    DEVSUPFUN special_linconv;
} devAiPrngAsync = {
    6, /* space for 6 functions */
    NULL,
    NULL,
    init_record,
    NULL,
    read_ai,
    NULL
};

epicsExportAddress(dset,devAiPrngAsync); /* change name */

void prng_cb(CALLBACK* cb);
```

Note the addition to the *prngState* struct of a CALLBACK.

```
static long init_record(aiRecord *pao)
{
    struct prngState* priv;
    unsigned long start;

    priv=malloc(sizeof(struct prngState));
    if(!priv){
        recGblRecordError(S_db_noMemory, (void*)pao,
            "devAoTimebase failed to allocate private struct");
        return S_db_noMemory;
    }

    /* New */
```

```

callbackSetCallback(prng_cb, &priv->cb);
callbackSetPriority(priorityLow, &priv->cb);
callbackSetUser(pao, &priv->cb);
priv->cb.timer=NULL;

recGblInitConstantLink(&pao->inp, DBF_ULONG, &start);

priv->seed=start;
pao->dpvt=priv;

return 0;
}

```

The callback function and priority are set.

```

static long read_ai(aiRecord *pao)
{
    struct prngState* priv=pao->dpvt;

    if( ! pao->pact ){
        /* start async operation */
        pao->pact=TRUE;
        callbackSetUser(pao, &priv->cb);
        callbackRequestDelayed(&priv->cb, 0.1);
        return 0;
    }else{
        /* complete operation */
        pao->pact=FALSE;
        return 0;
    }
}

```

The operation of *read_ai* changes substantially. When the record is processed control passes into *read_ai* with the *PACT* field set to false. To start an asynchronous operation this field is set to TRUE, and a delayed action is scheduled. While *PACT* is set the IOC will not try to process this record again.

When the callback has completed it must manually process the record which will complete the operation and allow *PACT* to be cleared.

```

void prng_cb(CALLBACK* cb)
{
    aiRecord* prec;
    struct prngState* priv;
    struct rset* prset;
    epicsInt32 raw;

    callbackGetUser(prec, cb);
    prset=(struct rset*)prec->rset;
    priv=prec->dpvt;

    raw=rand_r(&priv->seed);

    dbScanLock((dbCommon*)prec);
    prec->rval=raw;
    (*prset->process)(prec);
    dbScanUnLock((dbCommon*)prec);
}

```

This generic callback is taken from the *EPICS Application Developer's Guide* [\[AppDev\]](#). It manually invoke record processing. It this way *read_ai* is called while *PACT* is set.

Note: Due to the way database processing and the *PACT* flag are handled no additional locking is required. More complicated scenarios involving multiple PVs might require, for example, a mutex to guard *priv*→*seed*.

Remember to add *devprngasync.c* to *prngApp/src/Makefile*.

4.1 Running

The database file can be reused so only one change is necessary.

In *iocBoot/iocprng1/st.cmd* add a line:

```
< envPaths
cd ${TOP}
dbLoadDatabase "dbd/prng.dbd"
prng_registerRecordDeviceDriver pdbbase
dbLoadRecords("db/prng.db", "P=test:prng,D=Random,S=324235")
# V Add this line V
dbLoadRecords("db/prng.db", "P=test:prngasync,D=Random Async,S=324235")
cd ${TOP}/iocBoot/${IOC}
iocInit
```

5 Now What

Notice that the two seed values are the same and observe the values of the two PVs at run time. Now change the delay on the asynchronous callback (ie *callbackRequestDelayed()*) from 0.1 to 1.1 and compare the results.

These examples demonstrates writing device support for a single AI record. Support for other record types (BO, STRINGIN, EVENT, ...) differs only in which fields the *read_** function must update. Also interesting are some of the other fields which effect conversion of raw values in the analog record types, and the ability to bypass this conversion and specify value in engineering units directly.

6 References

- [1] [RecRef] The Epics Collaboration EPICS 3.14 Record Reference Manual Wiki http://www.aps.anl.gov/epics/wiki/index.php/RRM_3-14
- [2] [AppDev] Marty Kraimer et al. EPICS Application Developer's Guide . <http://www.aps.anl.gov/epics/base/R3-14/10-docs/AppDevGuide.pdf>